JEGYZŐKÖNYV

Adatkezelés XML környezetben

Féléves feladat: Sport Club

Készítette: **Takács Ákos**
Neptunkód: **VTJ4ES**
Dátum: **2025. 12. 03.**

Miskolc, 2025

Contents

Chapter 1

A feladat leírása

A feladat egy sport club (sportegyesület) adatainak modellezése és XML alapú megvalósítása. A rendszerben megjelennek az egyesületi sportolók, az edzők, az edzéscsoportok, a sportlétesítmények, valamint a tagsági viszonyok és versenyek. A cél egy olyan XML alapú adatleírás kialakítása, amelyre XSD séma épül, és amely később Java DOM API segítségével olvasható, módosítható, lekérdezhető és fájlba kiírható.

A tervezés és a megvalósítás során az angol nyelvű elnevezéseket használtam (entitásnevek, elemek, attribútumok), mivel ez a gyakorlat elterjedt a szoftverfejlesztésben.

A végleges modellben az alábbi fő entitások jelentek meg:

- **Athlete**: sportolók,
- **Membership**: tagságtípusok,
- **Coach**: edzők,
- **Location**: helyszínek,
- **TrainingGroup**: edzéscsoportok,
- **Competition**: versenyek.

A gyökérből kiinduló központi logikai entitás a **Sport_Club_VTJ4ES** dokumentum, amely tartalmazza a felsorolt entitások példányait, valamint a köztük lévő kapcsolat-elemeket (HasMembership, JoinGroup, LeadGroup, TakePlaceAt, CompeteIn).

Chapter 2

I. feladat – XML/XSD létrehozás

2.1 A feladat ER modellje

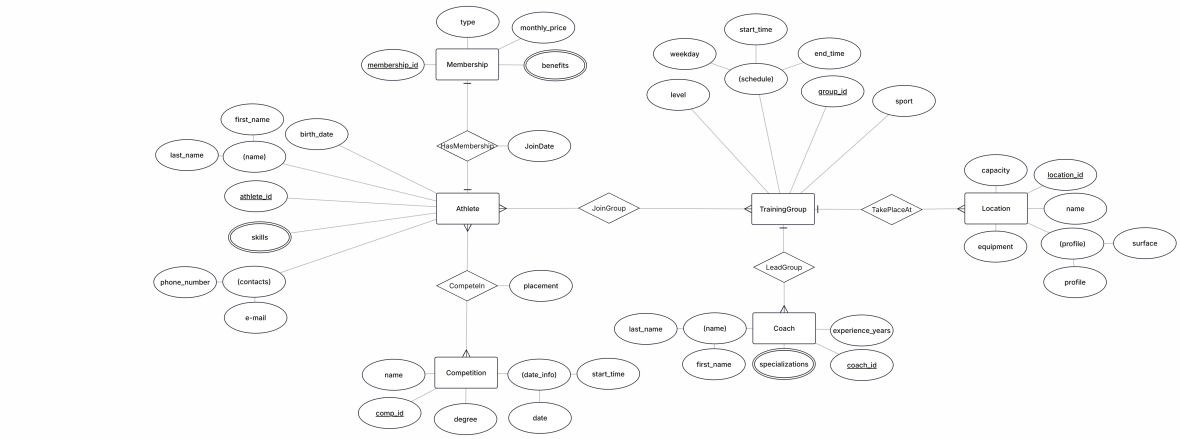

Figure 2.1: A sport club ER modellje

2.2 A feladat XDM modellje

Az XDM modell kialakításakor az ER modellben definiált entitásokat és kapcsolatokat kellett átvezetni XML-re. Figyelembe vettem az `1:1`, `1:N` és `N:M` kapcsolatokat, valamint az entitások elsődleges kulcsait.

Az `1:N` kapcsolatoknál a szaggatott nyíl mindig a „több” oldalon lévő kulcshoz mutat. Például a **Location** és **TrainingGroup** közötti kapcsolat esetén a `groupRef` attribútum mutat vissza a csoport elsődleges kulcsára. Az `N:M` kapcsolatok (JoinGroup, CompeteIn) esetében külön XDM kapcsolat-elem jelenik meg, amely tartalmazza a saját azonosítóját és a hivatkozásokat mindkét fő entitásra.

Többértékű tulajdonságok (pl. `skills`, `benefits`, `equipment`, `specializations`) XML-ben belső, ismétlődő elemekkel jelennek meg. A XDM modell gyökéreleme: **Sport_Club_VTJ4ES**.

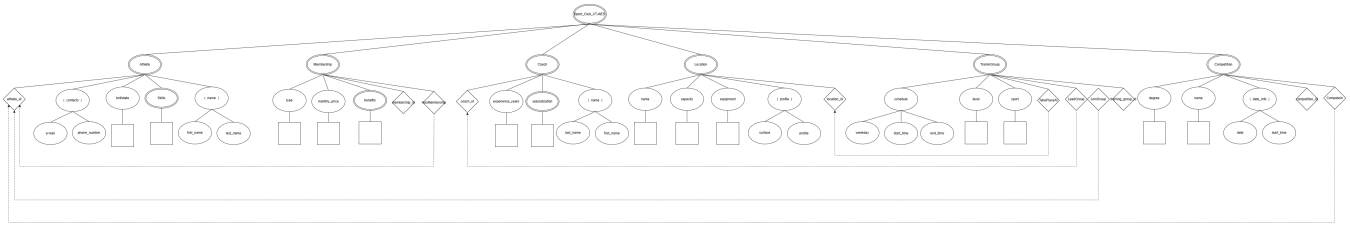

Figure 2.2: A sport club XDM modellje

2.3 Az XML dokumentum tervezése és példák

Az XML-dokumentum a sportegyesület XDM-modelljének megfelelően készült, ahol minden fő entitás (Athlete, Membership, Coach, Location, TrainingGroup, Competition) önálló XML-elemként jelenik meg, az azonosítók pedig attribútumként szerepelnek. Az összetett attribútumok (például név és elérhetőség) külön gyermekelemekben helyezkednek el, míg a többértékű tulajdonságok (skills, benefits, equipment, specializations) ismétlődő elemek formájában jelennek meg. A kapcsolatok külön kapcsolat-elemekkel kerülnek leírásra (HasMembership, JoinGroup, LeadGroup, TakePlaceAt, CompeteIn), amelyek az érintett entitások azonosítóira hivatkozó `Ref` attribútumokat tartalmaznak, így egyértelműen modellezik az 1–N és N–M kapcsolatokat is.

Példa egy entitás XML-leképezésére (Athlete)

```
<Athlete athleteID="1">
    <name>
        <first_name>Daniel</first_name>
        <last_name>Kiss</last_name>
    </name>
    <birth_date>2001-04-10</birth_date>
    <contacts>
        <e_mail>daniel.kiss@example.com</e_mail>
        <phone_number>+36-30-111-1111</phone_number>
    </contacts>
```

```
    <skills>speed</skills>
    <skills>endurance</skills>
    <skills>technique</skills>
</Athlete>
```

Programkód 2.1: Athlete entitás XML-részlete

Az `Athlete` elem egy adatbázisbeli táblának felel meg, ahol az `athleteID` attribútum jelöli az elsődleges kulcsot. A név összetett attribútumként önálló `<name>` elemben szerepel, a keresztnév és vezetéknév külön gyermekelemekben. A többértékű `skills` attribútumot ismétlődő `<skills>` elemek reprezentálják, így egy sportolóhoz több készség is hozzárendelhető.

Példa 1–N kapcsolat XML-es ábrázolására (Athlete–Membership)

```
<HasMembership hasMembershipID="1" athleteRef="1" membershipRef="1"
   joinDate="2024-01-01"/>
<HasMembership hasMembershipID="2" athleteRef="2" membershipRef="1"
   joinDate="2023-09-15"/>
<HasMembership hasMembershipID="3" athleteRef="3" membershipRef="2"
   joinDate="2024-03-10"/>
```

Programkód 2.2: HasMembership kapcsolat (Athlete–Membership)

Az `HasMembership` elemek az Athlete és Membership entitások közötti 1–N kapcsolatot írják le. Az `athleteRef` és `membershipRef` attribútumok az érintett sportoló és tagság elsődleges kulcsaira hivatkoznak, míg a `joinDate` a belépés dátumát tárolja.

Példa N–M kapcsolat XML-es ábrázolására (Athlete–TrainingGroup)

```
<JoinGroup joinGroupID="1" athleteRef="1" groupRef="1"/>
<JoinGroup joinGroupID="2" athleteRef="1" groupRef="2"/>
<JoinGroup joinGroupID="3" athleteRef="2" groupRef="1"/>
<JoinGroup joinGroupID="4" athleteRef="3" groupRef="2"/>
```

Programkód 2.3: JoinGroup kapcsolat (Athlete–TrainingGroup)

A `JoinGroup` kapcsolat-elem az Athlete és TrainingGroup közötti N–M kapcsolatot modellezi. Az `athleteRef` és `groupRef` attribútumok az Athlete és TrainingGroup entitások azonosítóira mutatnak, így egy XML „kapcsolótáblát” kapunk, amelyben ugyanaz a sportoló több csoportban is szerepelhet, és egy csoporthoz több sportoló is tartozhat.

2.4 Az XML dokumentum alapján XMLSchema készítése

Az XSD séma a sportegyesület XML-adatmodelljének formális leírását adja, amely biztosítja az adatok szerkezetének, adattípusainak és kapcsolatainak érvényességét. Az egyszerű típusok (például `idType`, `moneyType`, `timeHMType`, `placementType`) segítségével biztosított az azonosítók, pénzösszegek, időformátumok és helyezések tartományellenőrzése. Ezekre épülnek a közös komplex típusok (`NameType`, `ContactsType`, `ProfileType`, `ScheduleType`), valamint az egyes entitástípusok (`AthleteType`, `MembershipType`, `CoachType`, `LocationType`, `TrainingGroupType`, `CompetitionType`). A kapcsolatokhoz külön komplex típusok készültek (`HasMembershipType`, `JoinGroupType`, `LeadGroupType`, `TakePlaceAtType`, `CompeteInType`), amelyek attribútumként tárolják az érintett entitások azonosítóira mutató hivatkozásokat. A `Sport_Club_VTJ4ES` gyökérelem alatt globális elemek sorozata jelenik meg, és `xs:key` / `xs:keyref` segítségével kerültek definiálásra az elsődleges kulcsok és idegen kulcsok, míg `xs:unique` elemek további egyediségi megszorításokat valósítanak meg.

Példa entitás XSD-leképezésére (AthleteType)

```
<xs:simpleType name="idType">
    <xs:restriction base="xs:integer">
        <xs:minInclusive value="1"/>
    </xs:restriction>
</xs:simpleType>

<xs:complexType name="NameType">
    <xs:sequence>
        <xs:element name="first_name" type="xs:string"/>
        <xs:element name="last_name" type="xs:string"/>
    </xs:sequence>
</xs:complexType>

<xs:complexType name="AthleteType">
    <xs:sequence>
        <xs:element name="name" type="NameType"/>
        <xs:element name="birth_date" type="xs:string"/>
        <xs:element name="contacts" type="ContactsType"/>
        <xs:element name="skills" type="xs:string" maxOccurs="unbounded"/>
    </xs:sequence>
    <xs:attribute name="athleteID" type="idType" use="required"/>
</xs:complexType>

<xs:element name="Athlete" type="AthleteType"/>
```

Programkód 2.4: AthleteType komplex típus XSD-ben

Az `idType` egyszerű típus biztosítja, hogy az azonosítók mindig 1-nél nagyobb egész számok legyenek. Az `AthleteType` komplex típus írja le a sportoló entitást: a gyermekelemek között szerepel a név, születési dátum, elérhetőségek és a többértékű `skills` elem (maxOccurs=`"unbounded"`). Az `athleteID` attribútum az entitás elsődleges kulcsát reprezentálja, míg a globális `Athlete` elem az XML-ben ténylegesen megjelenő elemtípust adja.

Kapcsolattípus és kulcsok XSD-ben (HasMembership, key, keyref, unique)

```
<xs:complexType name="HasMembershipType">
    <xs:attribute name="hasMembershipID" type="idType" use="required"/>
    <xs:attribute name="athleteRef" type="idType" use="required"/>
    <xs:attribute name="membershipRef" type="idType" use="required"/>
    <xs:attribute name="joinDate" type="xs:string" use="required"/>
</xs:complexType>

<xs:key name="athlete_key">
    <xs:selector xpath="Athlete"/>
    <xs:field xpath="@athleteID"/>
</xs:key>

<xs:key name="membership_key">
    <xs:selector xpath="Membership"/>
    <xs:field xpath="@membershipID"/>
</xs:key>

<xs:keyref name="hasMembership_athlete_fk" refer="athlete_key">
    <xs:selector xpath="HasMembership"/>
    <xs:field xpath="@athleteRef"/>
</xs:keyref>

<xs:keyref name="hasMembership_membership_fk" refer="membership_key">
    <xs:selector xpath="HasMembership"/>
    <xs:field xpath="@membershipRef"/>
</xs:keyref>

<xs:unique name="oneCoachPerGroup">
    <xs:selector xpath="LeadGroup"/>
    <xs:field xpath="@groupRef"/>
</xs:unique>

<xs:unique name="oneLocationPerGroup">
    <xs:selector xpath="TakePlaceAt"/>
    <xs:field xpath="@groupRef"/>
</xs:unique>
```

Programkód 2.5: Kapcsolatok és kulcsok XSD-ben

A `HasMembershipType` kapcsolat-típust modellez az Athlete és Membership entitások között, ahol az `athleteRef` és `membershipRef` attribútumok idegen kulcsként működnek. Az `xs:key` elemek az Athlete és Membership elsődleges kulcsait definiálják, míg az `xs:keyref` gondoskodik arról, hogy a hivatkozott azonosítók valóban létező entitásokra mutassanak. Az `xs:unique` elemek további üzleti szabályokat valósítanak meg, például azt, hogy egy csoporthoz legfeljebb egy edző (`LeadGroup`) vagy egy helyszín (`TakePlaceAt`) tartozhat.

Chapter 3

II. feladat – DOM alapú feldolgozás

3.1 DOM tervezése és általános működés

A DOM (Document Object Model) alapú megoldás célja, hogy a korábban megtervezett XML adatmodellt Java kódból kényelmesen lehessen bejárni, módosítani és kiírni. A DOM lényege, hogy a `Sport_Club_VTJ4ES.xml` dokumentum teljes tartalmát egy fa struktúraként betölti a memóriába: minden XML elemhez, attribútumhoz és szöveghez külön `Node` (illetve `Element`) objektum tartozik. Így a program a fa tetszőleges pontján szabadon tud:

- elemeket keresni (pl. `getElementsByTagName("Athlete")`),
- attribútumokat és szöveges tartalmakat kiolvasni vagy módosítani,
- új elemeket beszúrni vagy meglévőket törölni,
- a módosított DOM-fát új XML dokumentumként kiírni.

A DOM megközelítés előnye, hogy a teljes dokumentum egyszerre, strukturáltan elérhető, ezért ideális összetett lekérdezésekhez és többlépéses módosításokhoz. Hátránya, hogy nagyobb XML állomány esetén jelentős memóriát foglal. A gyakorlatban ez a feladat méreténél nem okoz gondot, viszont jól szemlélteti a fa alapú feldolgozás lépéseit.

A DOM megvalósítás négy fő Java osztályra épül:

- `VTJ4ESDomRead`: az XML beolvasása és bejárása,
- `VTJ4ESDomQuery`: lekérdezések futtatása a DOM-fán,
- `VTJ4ESDomWrite`: a DOM-fa kiírása formázott XML-be,
- `VTJ4ESDomModify`: az XML-tartalom módosítása DOM-on keresztül.

3.2 VTJ4ESDomRead – Adatolvasás DOM-mal

A `VTJ4ESDomRead` osztály egy DOM-alapú XML-olvasó program, amely a teljes XML dokumentumot memóriába tölti, majd külön metódusokkal (`readAthletes`, `readMemberships`, `readCoaches`, `readLocations`, `readTrainingGroups`, `readCompetitions`) kiolvassa az egyes entitások adatait és strukturált formában a konzolra írja.

DOM parser inicializálása

```
DocumentBuilderFactory factory = DocumentBuilderFactory.newInstance();
factory.setIgnoringElementContentWhitespace(true);
DocumentBuilder builder = factory.newDocumentBuilder();

Document document = builder.parse(new File("../VTJ4ES_XML.xml"));
document.getDocumentElement().normalize();
```

Programkód 3.1: DOM parser létrehozása és dokumentum betöltése

Athlete elemek feldolgozása

```
NodeList athleteList = document.getElementsByTagName("Athlete");
for (int i = 0; i < athleteList.getLength(); i++) {
    Node node = athleteList.item(i);
    if (node.getNodeType() == Node.ELEMENT_NODE) {
        Element e = (Element) node;

        String athleteId = e.getAttribute("athleteID");

        Element nameElem = (Element) e.getElementsByTagName("name").item(0);
        String firstName = nameElem.getElementsByTagName("first_name")
                                   .item(0).getTextContent();
        String lastName = nameElem.getElementsByTagName("last_name")
                                  .item(0).getTextContent();
    }
}
```

Programkód 3.2: Athlete elemek bejárása VTJ4ESDomRead-ban

Többértékű skills elemek kezelése

```
NodeList skills = e.getElementsByTagName("skills");
for (int j = 0; j < skills.getLength(); j++) {
    String skill = skills.item(j).getTextContent();
```

```
    // kiiras vagy tovabbfeldolgozas
}
```

Programkód 3.3: Többértékű skills elemek kezelése

3.3 VTJ4ESDomQuery – Adatlekérdezés DOM-mal

A `VTJ4ESDomQuery` osztály DOM-alapú lekérdezéseket valósít meg: szűri a sportolókat bizonyos skillek alapján, megkeresi például a legnagyobb kapacitású helyszínt, vagy lekérdezi a 2000 után született sportolókat. Az eredményeket `StringBuilder` segítségével XML-szerű kimenetté állítja össze.

Sportolók szűrése skill alapján

```
NodeList athleteList = doc.getElementsByTagName("Athlete");
for (int i = 0; i < athleteList.getLength(); i++) {
    Node node = athleteList.item(i);
    if (node.getNodeType() == Node.ELEMENT_NODE) {
        Element athlete = (Element) node;

        NodeList skills = athlete.getElementsByTagName("skills");
        boolean hasSpeed = false;
        for (int j = 0; j < skills.getLength(); j++) {
            String skillValue = skills.item(j).getTextContent();
            if ("speed".equalsIgnoreCase(skillValue)) {
                hasSpeed = true;
                break;
            }
        }

        if (hasSpeed) {
            String athleteId = athlete.getAttribute("athleteID");
            // eredmeny osszegyujtese StringBuilder-be
        }
    }
}
```

Programkód 3.4: Sportolók szűrése skill alapján

Legnagyobb kapacitású Location kiválasztása

```
NodeList locationList = doc.getElementsByTagName("Location");
int maxCapacity = -1;
```

```
Element maxLocation = null;

for (int i = 0; i < locationList.getLength(); i++) {
    Node node = locationList.item(i);
    if (node.getNodeType() == Node.ELEMENT_NODE) {
        Element location = (Element) node;
        String capacityText = location.getElementsByTagName("capacity")
                                      .item(0).getTextContent();
        int capacity = 0;
        try {
            capacity = Integer.parseInt(capacityText.trim());
        } catch (NumberFormatException e) {
            // ervenytelen ertek figyelmen kivul hagyasa
        }

        if (capacity > maxCapacity) {
            maxCapacity = capacity;
            maxLocation = location;
        }
    }
}
```

Programkód 3.5: Legnagyobb kapacitású helyszín (Location) kiválasztása

3.4 VTJ4ESDomWrite – Adatírás DOM-ból XML-be

A `VTJ4ESDomWrite` osztály a memóriában lévő DOM fát írja ki formázott XML dokumentumba. A `Transformer` API segítségével gondoskodik az olvasható, behúzásokkal ellátott kimenetről, mind a konzolra, mind fájlba.

```
TransformerFactory transformerFactory = TransformerFactory.newInstance();
Transformer transformer = transformerFactory.newTransformer();

transformer.setOutputProperty(OutputKeys.INDENT, "yes");
transformer.setOutputProperty("{http://xml.apache.org/xslt}indent-amount",
    "2");
transformer.setOutputProperty(OutputKeys.ENCODING, "UTF-8");

DOMSource source = new DOMSource(doc);
StreamResult consoleResult = new StreamResult(System.out);
transformer.transform(source, consoleResult);
```

Programkód 3.6: DOM dokumentum kiírása a konzolra

```
private static void writeDocumentToFile(Document doc, String filename)
        throws TransformerException {
```

```
    TransformerFactory transformerFactory = TransformerFactory.newInstance();
    Transformer transformer = transformerFactory.newTransformer();

    transformer.setOutputProperty(OutputKeys.INDENT, "yes");
    transformer.setOutputProperty("{http://xml.apache.org/xslt}indent-amount",
   "2");
    transformer.setOutputProperty(OutputKeys.ENCODING, "UTF-8");

    DOMSource source = new DOMSource(doc);
    StreamResult result = new StreamResult(new File(filename));
    transformer.transform(source, result);
}
```

Programkód 3.7: DOM dokumentum fájlba írása

3.5 VTJ4ESDomModify – Adatmódosítás DOM-mal

A `VTJ4ESDomModify` osztály a beolvasott DOM-fát módosítja: például előtagot fűz az `athleteID` értékekhez, módosítja a tagsági díjakat, minimális kapacitást állít be a helyszíneknél, illetve jelölést ad a `profile_text` végére. A módosítások után a frissített dokumentum formázott XML-ként kerül kiírásra.

Módosítások végrehajtása és kiírása

```
Document doc = dBuilder.parse(xmlFile);
doc.getDocumentElement().normalize();

modifyAthletes(doc);
modifyMemberships(doc);
modifyLocations(doc);

TransformerFactory transformerFactory = TransformerFactory.newInstance();
Transformer transformer = transformerFactory.newTransformer();

transformer.setOutputProperty(OutputKeys.INDENT, "yes");
transformer.setOutputProperty("{http://xml.apache.org/xslt}indent-amount",
   "2");
transformer.setOutputProperty(OutputKeys.ENCODING, "UTF-8");

DOMSource source = new DOMSource(doc);
StreamResult consoleResult = new StreamResult(System.out);
transformer.transform(source, consoleResult);
```

Programkód 3.8: Módosítások végrehajtása és kiírása

Athlete azonosítók módosítása

```
private static void modifyAthletes(Document doc) {
    NodeList athleteList = doc.getElementsByTagName("Athlete");

    for (int i = 0; i < athleteList.getLength(); i++) {
        Node node = athleteList.item(i);
        if (node.getNodeType() == Node.ELEMENT_NODE) {
            Element athlete = (Element) node;
            String id = athlete.getAttribute("athleteID");

            if (id != null && !id.startsWith("ATH_")) {
                athlete.setAttribute("athleteID", "ATH_" + id);
            }
        }
    }
}
```

Programkód 3.9: Athlete ID-k módosítása VTJ4ESDomModify-ban

Location kapacitások és profil szöveg módosítása

```
private static void modifyLocations(Document doc) {
    NodeList locationList = doc.getElementsByTagName("Location");

    for (int i = 0; i < locationList.getLength(); i++) {
        Node node = locationList.item(i);
        if (node.getNodeType() == Node.ELEMENT_NODE) {
            Element location = (Element) node;

            Node capacityNode =
   location.getElementsByTagName("capacity").item(0);
            if (capacityNode != null && capacityNode.getNodeType() ==
   Node.ELEMENT_NODE) {
                String capacityText = capacityNode.getTextContent().trim();
                try {
                    int capacity = Integer.parseInt(capacityText);
                    if (capacity < 100) {
                        capacity = 100;
                    }
                    capacityNode.setTextContent(String.valueOf(capacity));
                } catch (NumberFormatException e) {
                    // ervenytelen ertek figyelmen kivul hagyasa
                }
            }

            NodeList profileList = location.getElementsByTagName("profile");
```

```
24              if (profileList.getLength() > 0) {
25                  Element profile = (Element) profileList.item(0);
26                  Node profileTextNode =
    profile.getElementsByTagName("profile_text")
27                                                  .item(0);
28                  if (profileTextNode != null
29                          && profileTextNode.getNodeType() == Node.ELEMENT_NODE)
    {
30                      String oldText = profileTextNode.getTextContent();
31                      if (!oldText.endsWith(" (updated)")) {
32                          profileTextNode.setTextContent(oldText + " (updated)");
33                      }
34                  }
35              }
36          }
37      }
38  }
```

Programkód 3.10: Location elemek módosítása VTJ4ESDomModify-ban